

MICRO

SOFT

MICRO

SOFT

MICRO

SOFT

MICRO

SOFT

MICRO

SOFT

MICRO

SOFT

MICRO

SOFT

MICRO

SOFT

MICRO

SOFT

BASIC FOR KIM



From the Authors of KIM-1 BASIC

Before a computer can perform any useful function, it must be "told" what to do. Unfortunately, at this time, computers are not capable of understanding English or any other "human" language. This is primarily because our languages are rich with ambiguities and implied meanings. The computer must be told precise instructions and the exact sequence of operations to be performed in order to accomplish any specific task. Therefore, in order to facilitate human communication with a computer, programming languages have been developed.

KIM-1 BASIC* is a programming language both easily understood and simple to use. It serves as an excellent "tool" for applications in areas such as business, science and education. With only a few hours of using BASIC, you will find that you can already write programs with an ease that few other computer languages can duplicate.

Originally developed at Dartmouth University, BASIC language has found wide acceptance in the computer field. Although it is one of the simplest computer languages to use, it is very powerful. BASIC uses a small set of common English words as its "commands". Designed specifically as an "interactive" language, you can give a command such as "PRINT 2+2", and KIM-1 BASIC will immediately reply with "4". It isn't necessary to submit a card deck with your program on it and then wait hours for the results. Instead the full power of the KIM-1 is "at your fingertips".

Generally, if the computer does not solve a particular problem the way you expected it to, there is a "Bug" or error in your program, or else there is an error in the data which the program used to calculate its answer. If you encounter any errors in BASIC itself, please let us know and we'll see that it's corrected. Write a letter to us containing the following information:

- 1) System Configuration
- 2) Version of BASIC
- 3) A detailed description of the error. Include all pertinent information such as a listing of the program in which the error occurred, the data placed into the program and BASIC printout.

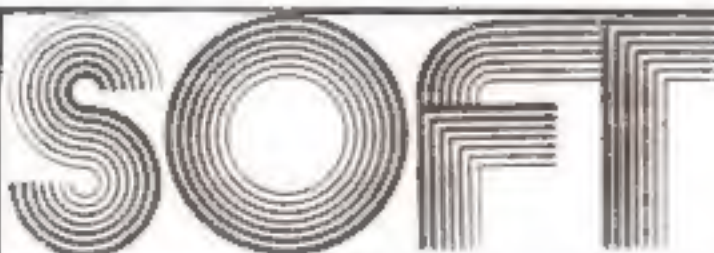
All of the information listed above will be necessary in order to properly evaluate the problem and correct it as quickly as possible. We wish to maintain as high a level of quality as possible with all of our KIM-1 software.

NOTE: BASIC is available under license or purchase agreements. Copying or otherwise distributing Microsoft software outside the terms of such an agreement may be a violation of copyright laws or the agreement itself.

If any immediate problems with Microsoft software are encountered, feel free to give us a call at 216/725-4568/4560

We hope that you enjoy KIM-1 BASIC, and are successful in using it to solve all of your programming needs.

* KIM-1 is a registered Trademark of MOS TECHNOLOGY
BASIC is a registered trademark of Dartmouth University



Distributed by:
Johnson Computer
P.O. Box 523
Medina, OH 44256
216/725-4568/4560

INTRODUCTION

We recommend that you try each example in this section as it is presented. This will enhance your "feel" for BASIC and how it is used.

Once your I/O device has typed "OK", you are ready to use KIM-1 BASIC.

NOTE: All commands to KIM-1 BASIC should end with a carriage return. The carriage return tells BASIC that you have finished typing the command. If you make a typing error, type a back-arrow (+), usually shift/0 or an underline, to eliminate the last character. Repeated use of "+" will eliminate previous characters. An at-sign (@) will eliminate the entire line that you are typing.

Now, try typing in the following:

```
PRINT 10/4 (end with carriage return)
```

KIM-1 BASIC will immediately print:

```
6
```

```
OK
```

The print statement you typed in was executed as soon as you hit the carriage return key. BASIC evaluated the formula after the "PRINT" and then typed out its value, in this case 6.

Now try typing in this:

```
PRINT 1/2,3*10 (" * " means multiply, " / " means divide)
```

BASIC will print:

```
.5      30
```

As you can see, KIM-1 BASIC can do division and multiplication as well as subtraction. Note how a ", " (comma) was used in the print command to print two values instead of just one. The comma divides the 72 character line into 5 columns, each 14 characters wide. The last two of the positions on the line are not used. The result is a ", " causes BASIC to skip to the next 14 column field on the terminal, where the value 30 was printed.

Commands such as the "PRINT" statements you have just typed in are called Direct Commands. There is another type of command called an Indirect Command. Every Indirect command begins with a Line Number. A Line Number is any integer from 0 to 64000.

Try typing in the following lines:

```
10 PRINT 2+3  
20 PRINT 2-3
```

A sequence of Indirect Commands is called a "Program". Instead of executing indirect statements immediately, KIM-1 BASIC saves Indirect Commands in the KIM-1's memory. When you type in RUN, BASIC will first execute the lowest numbered indirect statement that has been typed in, then the next highest, etc. for as many as were typed in.

Suppose we type in RUN now:

```
RUN
```

KIM-1 BASIC will type out:

```
5
```

```
-1
```

```
OK
```

In the example above, we typed in line 10 first and line 20 second. However, it makes no difference in what order you type in indirect statements, BASIC always puts them into correct numerical order according to the Line Number.

If we want a listing of the complete program currently in memory, we type in LIST . Type this in:

LIST

KIM-1 BASIC will reply with:

10 PRINT 2+3
20 PRINT 2-3
OK

Sometimes it is desirable to delete a line of a program altogether. This is accomplished by typing the Line Number of the line we wish to delete, followed only by a carriage return.

Type in the following:

10
LIST

KIM-1 BASIC will reply with:

20 PRINT 2-3
OK

We have now deleted line 10 from the program. There is only one way to get it back. To insert a new line 10, type in 10 followed by the statement we want BASIC to execute.

Type in the following:

10 PRINT 2*3
LIST

KIM-1 BASIC will reply with:

10 PRINT 2*3
20 PRINT 2-3
OK

There is an easier way to replace line 10 than deleting it and then inserting a new line. You can do this by just typing the new line 10 and hitting the carriage return. BASIC throws away the old line 10 and replaces it with the new one.

Type in the following:

10 PRINT 3-3
LIST

KIM-1 BASIC will reply with:

10 PRINT 3-3
20 PRINT 2-3
OK

It is not recommended that lines be numbered consecutively by increments of one (e.g., 1, 2, 3, ...). It may become necessary to insert a new line between two existing lines. An increment of 10 between line numbers is generally sufficient.

If you want to erase the complete program currently stored in memory, type in "NEW". If you are finished running one program and are about to read in a new one, be sure to type in "NEW" first. This should be done in order to prevent a mixture of the old and new programs.

Type in the following:

NEW

KIM-1 BASIC will reply with:

OK

Now type in:

LIST

KIM-1 BASIC will reply with:

OK

COMPANION

COMPANION TO THE SCHAUM'S OUTLINE SERIES' PROGRAMMING WITH BASIC

SECTION	PAGE	COMMENT
1.2	4	Skip to page 6, just after example 1.4. Timeshare and KIM BASIC share the interactive capability.
1.3	6	Rule 1: KIM allows multiple statements per line if each statement is separated from any previous statement of the same line by a colon (:).
2.3	14	For KIM, each NUMERIC VARIABLE must consist of a letter, a letter followed by an integer, or a letter followed by a letter. Similar conditions apply to a STRING VARIABLE with all STRING VARIABLES followed by a dollar sign (\$). FNS is a specific letter-letter-\$ that is not a usable string variable name on KIM.
2.7	17	Rule 3, Ex. 2.9 - Raising a number to the 1/2 or .5 power will result in the square root of that number. Similarly, to obtain the cube root of a number, raise it to the 1/3 or .333333 power. Again, the fourth root of a number is the same as that number raised to the 1/4 or .25 power.
2.8	18	Ex. 2.13 - The first, third and fourth statements are not valid with KIM. Multiple assignments are not valid. NOTE: On KIM, the word LET is optional.
2.9	18	Rule 4 - Also strings containing " " %.
2.11	22	The END statement is optional with KIM. If there is an END statement it does not have to be at the end of the program but can be embedded in the program.
2.13	24	Ex. 2.28 - Invalid on KIM. Instead use: 40 LET X1=(1.8-R)/(2*A) REM CALCULATE FIRST ROOT 50 LET X2=(1.8-R)/(2*A) REM CALCULATE SECOND ROOT Where the word LET is optional.
3.2	38	Does not pertain to KIM BASIC.
3.3	39	KIM has the prompting word "OK" rather than the symbol " ".
3.4	40	To delete characters with KIM type an underscore(_) or back going arrow (←). Type it once and one previous character is deleted. Type it twice and two previous characters are deleted, and so on.
3.5	41	See this manual for KIM system commands and how to use them.
3.6	43	Does not pertain to KIM.
4.1	53	See paragraph preceding Ex. 4.2 - KIM BASIC does not ignore trailing blanks.
4.2	54	See paragraph 2 - KIM BASIC also allows GOTO rather than THEN.
4.3	56	See Ex. 4.8 - THEN cannot replace GOTO in ON-GOTO statement.
4.4	58	Again, END is optional for KIM and can appear anywhere than the physical end of the program.
4.5	63	Only STEP with KIM, not BY.
4.6	63	Running variable after NEXT is optional when obvious.
	64	KIM loops will execute once under conditions 3(a), 3(b), and 3(c).
	64	See Ex. 4.15 - To effect indenting on KIM start the line with a colon (:): e.g. 165: LET Z=X+Y.
	70	See FIG. 4.10 - On printout, when I=3 and F=2 the fibonacci number will not be labeled prime. On line 140, when I=3, J=2 and F=2. The BASIC used by Mr. Gottfried does not execute the loop at all (see pg. 64 top) and skips down to line 180. But KIM executes the loop once in which case line 160 is true and the program branches to line 190. For the program to work properly do the following: 105 ? "I=";I;"F=";F;"(PRIME)" 110 FOR I=4 TO N REM GENERATE FIBONACCI NUMBERS
5.1	81	See table 5.1 - KIM does not utilize COT. Use 1/TAN.
	81	See paragraph 1 - On KIM, negative numbers with functions that require positive arguments will generate an error message and the function will not be analyzed.
	83	See program listing. Change line 70 and add 72 and 74: 70: ? TAB(37); TAN(X); 72: IF X<0 THEN ? TAB(49);LOG(X);TAB(61);EXP(X); 74: ? REM PROVIDES CARRIAGE RETURN This is needed because the first value of X is not a valid argument for the LOG or EXP functions.
5.2	84	See first sentence - With KIM, arrays can be labeled letter-integer-\$ or letter-letter-\$.
5.6	97	RESTORE, but not RESTORES or RESTORE*, is valid with KIM.
5.1	113	For KIM: Functions must appear in the program before the line on which they are used. They cannot be grouped at the end of a program but should be grouped near the beginning. Also, no string functions, no multiple line functions, no multiple argument functions, and no function without an argument (Use a place holder variable e.g. 10 DEF FNK=C*2*B becomes 10 DEF FNK(X)=C*2*B where the argument of FNK is a variable that does not appear in the definition).

6.3	118	Does not pertain to KIM BASIC.
6.4	121	Not applicable to KIM.
6.5	123	When using ASC, any letter argument must appear in quotes: 50 C=ASC("P").
	123	CHR\$ is a string function which returns a one character string which contains the ASCII equivalent of the argument. ASC takes the first character of a string and converts it to its ASCII decimal value. One of the most common uses of CHR\$ is to send a special character to the user's terminal. The most often used of these characters is the BEL (ASCII 7). Printing this character will cause a bell to ring on some terminals and a "beep" on many CRT's. This may be used as a preface to an error message, as a novelty, or just to wake up the user if he has fallen asleep. (Example: PRINT CHR\$(7)). A major use of special characters is on those CRT's that have cursor positioning and other special functions (such as turning on a hard copy printer). As an example, try sending a form feed (CHR\$(12)) to your CRT. On most CRT's this will usually cause the screen to erase and the cursor to "home" or move to the upper left corner. Some CRT's give the user the capability of drawing graphs and curves in a special point-plotter mode. This feature may easily be taken advantage of through use of KIM-1 BASIC's CHR\$ function.
	123	Example 6.13, BASIC line number 70 - should read 70 IF L10=ASC(" ") THEN 110
	123	Ex. 6.15 uses CHANGE which is not a KIM function.
	126	Example 6.15, Program line 230 - not valid for KIM BASIC. Use 230 FOR I=1 TO LEN(NS)-L10:ASC(MID\$(NS,I,1)):NEXT
6.6	127	See EX. 6.16 and preceding paragraph - With KIM, an argument is needed for the function RND.
6.7	127	The necessary argument for RND affects the random number generator and is sometimes called the "seed." e.g. 50 INPUT "PLEASE ENTER RANDOM SEED "RS 60 X=RND(RS)
	128	See Ex. 6.2, the program outline, point 21c) - A carriage return only in response to a question will end execution of the KIM program. Use space then carriage return.
	130	See Ex. 6.2, Fig. 6.13 - Program uses multiline function and is therefore not compatible with KIM BASIC. Use the following 15 DEF FNK(X)=(1+INT(6*RND(RS)))>1+INT(6*RND(RS)) 20 INPUT "RANDOM SEED"RS 90 K=FNK(1) 180 K=FNK(1)
6.9	137	Line 720 of program should read as indicated by flowchart: 720 T3=T16/J1*IP2/C16/J1 After leaving a FOR-NEXT LOOP such as in lines 660-710 the value of the running variable is one more than the specified final index value, i.e. it leaves the 660-710 FOR-NEXT LOOP as I=7 and not 6.
	138	Although Mr. Gorfried did not use quotations around his response to NAME\$, it is necessary to do so on KIM because of the comma used in last-name-first entries. See section 2.9, Page 18, rule #4 and Ex. 2.15. In order to avoid having to use quotation marks change line 210 to: 210 INPUT N2\$,N1\$ Where N2\$ is the last name and N1\$ is the first name or beginning initials. (See again section 2.9, page 18 regards multiple inputs). Note that here one might have been tempted to use FNS for First Name but the specific character combination FNS is not acceptable by KIM because it too closely resembles the label of a user defined function.
6.10	143	On a 20K computer system with 8K BASIC it may be necessary to dispense with the blank lines, the remarks, and perhaps even lines 560 through 610 in order to have enough room to run the program.

Chapter 7 does not pertain to KIM.

Chapter 8 does not pertain to KIM.

DICTIONARY

Note: In the following an argument of V or W denotes a numeric variable, X denotes a numeric expression, X\$ denotes a string expression and an I or J denotes an expression that is truncated to an integer before the statement is executed. Truncation means that any fractional part of the number is lost, e.g. 3.9 becomes 3, 4.01 becomes 4.

An expression is a series of variables, operators, function calls and constants which after the operations and function calls are performed using the precedence rules, evaluates to a numeric or string value.

A constant is either a number (3.14) or a string literal ("FCO").

AND 2 IF A > 5 AND B > 2 THEN 7 If expression 1 (A > 5) AND expression 2 (B > 2) are both true, then branch to line 7.

ASC(X\$) Returns the ASCII numeric value of the first character of the string expression X\$. An FC error will occur if X\$ is the null string.

ATN(X) Gives the arctangent of the argument X. The result is returned in radians and ranges from $-\pi/2$ to $\pi/2$. ($\pi/2 = 1.5708$)

ABS(X) Gives the absolute value of the expression X.

CHR\$(i) Returns a one character string whose single character is the ASCII equivalent of the value of the argument (i) which must be ≥ 0 and $i \leq 255$.

CLEAR Clears all variables, resets FOR & GOSUB state and RESTORE data.

CONT Continues program execution after a control/c is typed or a STOP statement is executed. You cannot continue after any error, after modifying your program, or before your program has been run. One of the main purposes of CONT is debugging. Suppose at some point after running your program, nothing is printed. This may be because your program is performing some time consuming calculation, but it may be because you have fallen into an "infinite loop". An infinite loop is a series of BASIC statements from which there is no escape. The KIM-1 will keep executing the series of statements over and over, until you intervene or until power to the KIM-1 is cut off. If you suspect your program is in an infinite loop, type in a control/c. The line number of the statement BASIC was executing will be typed out. After BASIC has typed out OK, you can use PRINT to type out some of the values of your variables. After examining these values you may become satisfied that your program is functioning correctly. You should then type in CONT to continue executing your program where it left off, or type a direct GOTO statement to resume execution of the program at a different line. You could also use assignment (LET) statements to set some of your variables to different values. Remember, if you control/c a program and expect to continue it later, you must not get any errors or type in any new program lines. If you do, you won't be able to continue and will get a "CH" (continue not) error. It is impossible to continue a direct command. CONT always resumes execution at the next statement to be executed in your program when control/c was typed.

COS(X) Gives the cosine of the expression X. X is interpreted as being in radians.

DATA Specifies data, read from left to right. Information appears in data statements in the same order as it will be read in the program.

DEF 100 DEF FNA(V)=V/B+C The user can define functions like the built-in functions (SQR, SGN, ABS, etc.) through the use of the DEF statement. The name of the function is "FN" followed by any legal variable name, for example: FNX, FNJ7, FNK0, FNR2. User defined functions are restricted to one line. A function may be defined to be any expression, but may only have one argument. In the example B & C are variables that are used in the program. Executing the DEF statement defines the function. User defined functions can be redefined by executing another DEF statement for the same function. User defined string functions are not allowed. "V" is called the dummy variable. 110 Z=FNA(3) Execution of this statement following the above would cause Z to be set to 3/B+C, but the value of V would be unchanged.

DIM 113 DIM A(3), B(10) Allocates space for arrays. All array elements are set to zero by the DIM statement. 114 DIM R3(5,5), D3(2,2,2) Arrays can have more than one dimension. Up to 255 dimensions are allowed, but due to the restriction of 72 characters per line the practical maximum is about 34 dimensions. Arrays can be dimensioned dynamically during program execution. If an array is not explicitly dimensioned with a DIM statement, it is assumed to be a single dimensioned matrix of whose single subscript may range from 0 to 10 (eleven elements). 117 A(8)=4 If this

statement was encountered before a DIM statement for A was found in the program, it would be as if a DIM A(10) had been executed previous to the execution of line 117. All subscripts start at zero (0), which means that DIM X(100) really allocates 101 matrix elements.

END Terminates program execution without printing a BREAK message. (see STOP) CONT after an END statement causes execution to resume at the statement after the END statement. END can be used anywhere in the program, and is optional.

EXP(X) Gives the constant "E" (2.71828) raised to the power X. (E^X) The maximum argument that can be passed to EXP without overflow occurring is 87.3365.

FOR 300 FOR V=1 TO 9.3 STEP .6 (see NEXT statement) V is set equal to the value of the expression following the equal sign, in this case 1. This value is called the initial value. Then the statements between FOR and NEXT are executed. The final value is the value of the expression following the TO. The step is the value of the expression following STEP. When the NEXT statement is encountered, the step is added to the variable. 310 FOR V=1 TO 9.3 If no STEP was specified, it is assumed to be one. If the step is positive and the new value of the variable is \leq final value (9.3 in this example), or the step value is negative and the new value of the variable is \geq the final value, then the first statement following the FOR statement is executed. Otherwise, the statement following the NEXT statement is executed. All FOR loops execute the statements between the FOR and the NEXT at least once, even in cases like FOR V=1 TO 0. 315 FOR V=10*N TO 3.4/Q STEP SQR(R) Note that expressions (formulas) may be used for the initial, final and step values in a FOR loop. The values of the expressions are computed only once, before the body of the FOR...NEXT loop is executed. 320 FOR V=9 TO 1 STEP -1 When the statement after the NEXT is executed, the loop variable is never equal to the final value, but is equal to whatever value caused the FOR...NEXT loop to terminate. The statements between the FOR and its corresponding NEXT in both examples above (310 & 320) would be executed 9 times. 330 FOR W=1 TO 10: FOR W=1 TO 10: NEXT W: NEXT W Error! do not use nested FOR...NEXT loops with the same index variable. FOR loop nesting is limited only by the available memory.

FRE(X) OR FRE(X\$) 270 PRINT FRE(0) Gives the number of memory bytes currently unused by BASIC. Memory allocated for STRING space is not included in the count returned by FRE.

GOTO Branches to the statement specified.

GOSUB Branches to the specified statement until a RETURN is encountered; when a branch is then made to the statement after the GOSUB. GOSUB nesting is limited only by the available memory.

IF...GOTO Equivalent to IF...THEN, except that IF...GOTO must be followed by a line number.

IF...THEN IF X > 10 THEN 5 Branches to specified statement if the relation is true. 20 IF X < 0 THEN PRINT "X LESS THAN 0" Executes all of the statements on the remainder of the line after the THEN if the relation is true.

INPUT Requests data from the terminal (to be typed in). Each value must be separated from the preceding value by a comma (,). The last value typed should be followed by a carriage return. A "?" is typed as a prompt character. If more data was requested in an INPUT statement than was typed in, a "?" is printed and the rest of the data should be typed in. If more data was typed in than was requested, the extra data will be ignored. Strings must be input in the same format as they are specified in DATA statements.

5 INPUT "VALUE"; V Optionally types a prompt string ("VALUE") before requesting data from the terminal. If carriage return is typed to an input statement, BASIC returns to command mode. Typing CONT after an INPUT command has been interrupted will cause execution to resume at the INPUT statement.

INT(X) Returns the largest integer less than or equal to its argument X. For example: INT (.23) = 0, INT (7) = 7, INT

INT $(-1)=-1$, $\text{INT } (-2)=-2$, $\text{INT } (1.1)=1$.

The following would round X to D decimal places:
INT $(X*10^{(D+.5)})/10^{(D)}$

LEFTS(X\$, I) Gives the leftmost I characters of the string expression X\$. If I $(=0$ or >255) an FC error occurs.

LEN(X\$) Gives the length of the string expression X\$ in characters (bytes). Non-printing characters and blanks are counted as part of the length.

LET Assigns a value to a variable. "LET" is optional.

LIST **LIST 100- LIST 100 LIST 100-200** Lists current program optionally starting at specified line. List can be control-C'd (BASIC will finish listing the current line).

LOAD Loads the program from the cassette tape. A NEW command is automatically done before the LOAD command is executed. When done, the LOAD will type out OK as usual.

LOG(X) Gives the natural (Base E) logarithm of its argument X. To obtain the Base Y logarithm of X use the formula $\text{LOG}(X)/\text{LOG}(Y)$. Example: The base 10 (common) log of 7 = $\text{LOG}(7)/\text{LOG}(10)$.

MIDS(X\$, I) MIDS called with two arguments returns characters from the string expression X\$ starting at character position I. If I $>\text{LEN}(X\$)$, then MIDS returns a null (zero length) string. If I $(=0$ or $>255)$, an FC error occurs.

MIDS(X\$, I, J) MIDS called with three arguments returns a string expression composed of the characters of the string expression X\$ starting at the Ith character for J characters. If $J >\text{LEN}(X\$)$, MIDS returns a null string. If I or J $(=0$ or $>255)$, an FC error occurs. If J specifies more characters than are left in the string, all characters from the Ith on are returned.

NEW Deletes current program and all variables.

NEXT Marks the end of a FOR loop. If no variable is given, matches the most recent FOR loop. A single NEXT may be used to match multiple FOR statements. NEXT V,W is equivalent to NEXT V:NEXT W.

NOT **IF NOT Q3 THEN 4** If expression "NOT Q3" is true (because Q3 is false), then branch to line 4. Note: **NOT** $(1)=0$ (**NOT** true=false).

NULL **NULL 3** Sets the number of null (ASCII 0) characters printed after a carriage return/line feed. The number of nulls printed may be set from 0 to 71. This is a must for hardcopy terminals that require a delay after a carriage return/line feed. It is necessary to set the number of nulls typed on CRLF to 0 before a paper tape of a program is read in from a Teletype (TELETYPE is a registered trademark of the TELETYPE CORPORATION). Use the null command to set the number of nulls to zero. When you punch a paper tape of a program using the list command, null should be set $=3$ for 10 CPS terminals, $=6$ for 30 CPS terminals. When not making a tape, we recommend that you use a null setting of 0 or 1 for Teletypes, and 2 or 3 for hard copy 30 CPS terminals. A setting of 0 will work with Teletype compatible CRT's.

ON...GOSUB Identical to "ON...GOTO" except that a subroutine call (GOSUB) is executed instead of a GOTO. RETURN from the GOSUB branches to the statement after the ON...GOSUB.

ON...GOTO **100 ON I GOTO 10,20,30,40** Branches to the line indicated by the Ith number after the GOTO. That is: **IF I=1, THEN GOTO LINE 10 IF I=2, THEN GOTO LINE 20 IF I=3, THEN GOTO LINE 30 IF I=4, THEN GOTO LINE 40.** If I >0 or I attempts to select a non-existent line $I >5$ in this case), the statement after the ON statement is executed. However, if I is >255 or (0) , an FC error message will result. As many line numbers as will fit on a line can follow an ON...GOTO. **105 ON SGN(X)+2 GOTO 40,50,60** This statement will branch to line 40 if the expression X is less than zero, to line 50 if it equals zero, and to line 60 if it is greater than zero.

OR **IF A $(1$ OR B $(2$ THEN 2** If either expression 1 (A (1) OR expression 2 (B (2) is true, then branch to line 2.

PEEK(I) The PEEK function returns the contents of memory address I. The value returned will be $=$ 0 and <255 . If I is >65535 or (0) , an FC error will occur. An attempt to read a non-existent memory address will return an unknown value. (see POKE statement)

POKE **357 POKE I,J** The POKE statement stores the byte specified by its second argument (J) into the location given by its first argument (I). The byte to be stored must be

$=$ 0 and <255 , or an FC error will occur. The address (I) must be $=$ 0 and <65535 , or an FC error will result. Careless use of the POKE statement will probably cause you to "poke" BASIC to death; that is, the machine will hang, and you will have to reload BASIC and will lose any program you had typed in. A POKE to a non-existent memory location is harmless. One of the main uses of POKE is to pass arguments to machine language subroutines. You could also use PEEK and POKE to write a memory diagnostic or an assembler in BASIC.

POS(I) Gives the current position of the terminal print head (or cursor on CRT's). The leftmost character position on the terminal is position zero and the right most is 71.

PRINT Prints the value of expressions on the terminal. If the list of values to be printed out does not end with a comma (,) or a semicolon (;), then a carriage return/line feed is executed after all the values have been printed. Strings in quotes (") may also be printed. If a semicolon separates two expressions in the list, their values are printed next to each other. If a comma appears after an expression in the list, and the print head is at print position 56 or more, then a carriage return/line feed is executed. If the print head is before print position 56, then spaces are printed until the carriage is at the beginning of the next 14 column field (until the carriage is at column 14, 28, 42 or 56...), If there is no list of expressions to be printed, then a carriage return/line feed is executed. String expressions may be printed.

READ Reads data into specified variables from a DATA statement. The first piece of data read will be the first piece of data listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA statement, and so on. When all of the data have been read from the first DATA statement, the next piece of data to be read will be the first piece listed in the second DATA statement of the program. Attempting to read more data than there is in all the DATA statements in a program will cause an OD (out of data) error.

REM Allows the programmer to put comments in his program. REM statements are not executed, but can be branched to. A REM statement is terminated by end of line. But not by a ":",

RESTORE Allows the re-reading of DATA statements. After a RESTORE, the next piece of data read will be the first piece listed in the first DATA statement of the program. The second piece of data read will be the second piece listed in the first DATA statement, and so on as in a normal READ operation.

RETURN Causes a subroutine to return to the statement after the most recently executed GOSUB.

RIGHTS(X\$, I) Gives the rightmost I characters of the string expression X\$. When I $(=0$ or $>255)$ an FC error will occur. If I $>\text{LEN}(X\$)$ then RIGHTS returns all of X\$.

RND(I) Generates a random number between 0 and 1. The argument I controls the generation of random numbers as follows: X $(0$ starts a new sequence of random numbers using X. Calling RND with the same X starts the same random number sequence. X $=0$ gives the last random number generated. Repeated calls to **RND(0)** will always return the same random number. X >0 generates a new random number between 0 and 1.

Note that $(B-A)*\text{RND}(1)+A$ will generate a random number between A & B.

SAVE Saves on cassette tape the current program in the KIM's memory. The program in memory is left unchanged. More than one program may be stored on cassette using this command.

SGN(X) Gives 1 if X >0 , 0 if X $=0$ and -1 if X <0 .

SIN(X) Gives the sine of the expression X. X is interpreted as being in radians. Note: $\text{COS}(X)=\text{SIN}(X+3.14159/2)$ and $1 \text{ Radian} = 180/\pi \text{ degrees} = 57.2958 \text{ degrees}$, so that the sine of X degrees $=\text{SIN}(X)/57.2958$.

SPC(I) Prints I space (or blank) characters on the terminal. May be used only in a PRINT statement. X must be $=$ 0 and <255 or an FC error will result.

SQR(X) Gives the square root of the argument X. An FC error will occur if X is less than zero.

STOP Causes a program to stop execution and to enter command mode. **9000 STOP** Prints BREAK IN LINE 9000. (as per this example) CONT after a STOP branches to the statement following the STOP.

STR\$(X) Gives a string which is the character representation of the numeric expression X. For instance, **STR\$(3.1)="" 3.1"**.

TAB(I) **240 PRINT TAB(I)** Spaces to the specified print

position (column) on the terminal. May be used only in PRINT statements. Zero is the leftmost column on the terminal, 71 the rightmost. If the carriage is beyond position I, then no printing is done. I must be ≥ 0 and ≤ 255 .

TAN(X) Gives the tangent of the expression X. X is interpreted as being in radians.

USR(I) J=USR(T) Calls the user's machine language subroutine with the argument I. See POKE, PEEK and USR discussion.

VAL(X\$) Returns the string expression X\$ converted to a number. For instance, VAL ("3.14")=3.14. If the first non-space character of the string is not a plus (+) or minus (-) sign, a digit or a decimal point (.) then zero will be returned.

WAIT 805 WAIT I,J,K 806 WAIT I,J This statement reads the status of location I, exclusive OR's K with the status, and then AND's the result with J until a non-zero result is obtained. Execution of the program continues at the statement following the WAIT statement. If the WAIT statement only has two arguments, K is assumed to be zero. If you are waiting for a bit to become zero, there should be a one in the corresponding position of K. I, J and K must be ≥ 0 and ≤ 255 .

SYMBOLOLOGY AND SPECIAL KEYS

= Assigns a value to a variable. The LET is optional.

- Negation. Note that 0-A is subtraction, while -A is negation.

↑ (Usually a shift/P) Exponentiation $0 \uparrow 0 = 1$ 0 to any other power = 0. $A \uparrow B$, with A negative and B not an integer gives an FC error.

* Multiplication

/ Division

+ Addition. String Concatenation. The resulting string must be less than 256 characters in length or an LS error will occur.

- Subtraction

*) { < = > } String comparison operators. Comparison is made on the basis of ASCII codes, a character at a time until a difference is found. If during the comparison of two strings, the end of one is reached, the shorter string is considered smaller. Note that "A " is greater than "A" since trailing spaces are significant.

@ Erases current line being typed, and types a carriage return/line feed. An "@" is usually a shift/P.

← (backarrow or underline) Erases last character typed. If no more characters are left on the line, types a carriage return/line feed. "←" is usually a shift/D.

CARRIAGE RETURN A carriage return must end every line typed in. Returns print head or CRT cursor to the first position (leftmost) on line. A line feed is always executed after a carriage return.

CONTROL/C Interrupts execution of a program or a list command. Control/C has effect when a statement finishes execution, or in the case of interrupting a LIST command, when a complete line has finished printing. In both cases a return is made to BASIC's command level and OK is typed.

Prints "BREAK IN LINE XXXX", where XXXX is the line number of the next statement to be executed.

: [colon] A colon is used to separate statements on a line. Colons may be used in direct and indirect statements. The only limit on the number of statements per line is the line length. It is not possible to GOTO or GOSUB to the middle of a line.

CONTROL/O Typing a Control/O code causes BASIC to suppress all output until a return is made to command level, an input statement is encountered, another control/O is typed, or an error occurs.

? Question marks are equivalent to PRINT. For instance, $12=2$ is equivalent to PRINT $12=2$. Question marks can also be used in indirect statements. $10 ? X$, when listed will be typed as $10 PRINT X$.

ERROR MESSAGES

After an error occurs, BASIC returns to level and types OK. Variable values and the program text remain intact, but the program can not be continued and all GOSUB and FOR content is lost.

When an error occurs in a direct statement, no line number is printed.

Format of error messages:

Direct Statement ?XX ERROR

Indirect Statement ?XX ERROR IN YYYYY

In both of the above examples, "XX" will be the error code. The "YYYYY" will be the line number where the error occurred for the indirect statement.

The following are the possible error codes and their meanings:

BS Bad Subscript. An attempt was made to reference a array element which is outside the dimensions of the array. This error can occur if the wrong number of dimensions are used in a matrix reference; for instance, LET $A(2,1,1)=Z$ when A has been dimensioned DIM $A(2,2)$.

DD Double Dimension. After an array was dimensioned, another dimension statement for the same array was encountered. This error often occurs if an array has been given the default dimension 10 because a statement like $A(I)=3$ is encountered and then later in the program a DIM $A(100)$ is found.

FC Function Call error. The parameter passed to a math or string function was out of range. FC errors can occur due to: a) a negative array subscript (LET $A(-1)=0$). b) an unreasonably large array subscript ($1/32767$). c) LOG-negative or zero argument. d) SQR-negative argument. e) $A \uparrow B$ with A negative and B not an integer. f) a call to USR before the address of the machine language subroutine has been patched in. g) calls to MID\$, LEFT\$, RIGHT\$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC or ON. . . GOTO with an improper argument.

ID Illegal Direct. You cannot use an INPUT or DEFFN statement as a direct command.

NF NEXT without FOR. The variable in a NEXT statement corresponds to no previously executed FOR statement.

OD Out of Data. A READ statement was executed but all of the DATA statements in the program have already been read. The program tried to read too much data or insufficient data was included in the program.

OM Out of Memory. Program too large, too many variables, too many FOR loops, too many GOSUB's, too complicated an expression or any combination of the above.

OV Overflow. The result of a calculation was too large to be represented in BASIC's number format. If an underflow occurs, zero is given as the result and execution continues without any error message being printed.

SN Syntax error. Missing parenthesis in an expression, illegal character in a line, incorrect punctuation, etc.

RG RETURN without GOSUB. A RETURN statement was encountered without a previous GOSUB statement being executed.

US Undefined Statement. An attempt was made to GOTO, GOSUB or THEN to a statement which does not exist.

/0 Division by Zero.

- CN** Continue error. Attempt to continue a program when none exists, an error occurred, or after a new line was typed into the program.
- LS** Long String. Attempt was made by use of the concatenation operator to create a string more than 255 characters long.
- ST** String Temporaries. A string expression was too complex. Break it into two or more shorter ones.
- TM** Type Mismatch. The left hand side of an assignment statement was a numeric variable and the right hand side was a string, or vice versa.
- UF** Undefined Function. Reference was made to a user defined function which had never been defined.

Header string of messages: BASIC reserved messages: string: error

KB 3	KB 4	KB 9	PH
BAD SUBSCRIPT	BS	RECEIVED	PC
REDIMENS ONED ARRAY	DO	UNDEFINED STATEMENT	5
ILLEGAL QUANTITY	FC	DIVISION BY ZERO	0
ILLEGAL C OF T	FD	STRING TOO LONG	15
NEXT WITHOUT FOR	NF	FORMULA TOO COMPLEX	5
OUT OF DATA	OD	TYPE MISMATCH	TM
OUT OF MEMORY	OM	UNDEFINED FUNCTION	6
OVERFLOW	OV		
SYNTAX	SN		

ASCII CHARACTER CODES

DECIMAL	CHAR	DECIMAL	CHAR	DECIMAL	CHAR	DECIMAL	CHAR
000	NULL	033		065	A	097	a
001	SOH	034	^	066	B	098	b
002	STX	035	#	067	C	099	c
003	ETX	036	\$	068	D	100	d
004	EOT	037	%	069	E	101	e
005	ENQ	038	&	070	F	102	f
006	ACK	039		071	G	103	g
007	BEL	040		072	H	104	h
008	BS	041		073	I	105	
009	HT	042	*	074	J	106	j
010	LF	043	+	075	K	107	k
011	VT	044	,	076	L	108	l
012	FF	045	-	077	M	109	m
013	CR	046	.	078	N	110	n
014	SO	047	/	079	O	111	o
015	SI	048	0	080	P	112	p
016	DLE	049	1	081	Q	113	q
017	DC1	050	2	082	R	114	r
018	DC2	051	3	083	S	115	s
019	DC3	052	4	084	T	116	t
020	DC4	053	5	085	U	117	u
021	NAK	054	6	086	V	118	v
022	SYN	055	7	087	W	119	w
023	ETB	056	8	088	X	120	x
024	CAN	057	9	089	Y	121	y
025	EM	058	:	090	Z	122	z
026	SUB	059	:	091	[123	
027	ESCAPE	060	(092	\	124	
028	FS	061	=	093		125	
029	GS	062)	094	↑	126	~
030	RS	063	?	095	+	127	DEL
031	US	064	@	096			
032	SPACE						

LF = Line Feed

FF = Form Feed

CR = Carriage Return

DEL = Rubout

USAGE NOTES

INITIALIZATION DIALOG

STARTING BASIC After you have loaded BASIC, it will respond

MEMORY SIZE: If you type a 2 399 return, MEMORY SIZE: BASIC will use a near maximum memory upwards from 65536. That can find BASIC will stop saving when it finds one byte of RAM or swap space memory.

If you wish to allocate only part of the memory to BASIC, type the number of bytes of memory you wish to allocate in decimal. This might be done, for instance, if you were using part of the memory for a machine language subroutine.

There are 4096 bytes of memory in a 4K system, and 8192 bytes in an 8K system.

BASIC with: then 254

TERMINAL WIDTH This is to set the output line width for PRN7 printers only. Type in the number characters in the line width for the particular terminal or other output device you are using. This may be any number from 1 to 255, depending on the terminal. If no answer is given (i.e. a carriage return is typed) the line width is set to 72 characters.

Now the BASIC interpreter is going to allow a way to delete the lines in a file. The only left to now give the memory space to store your programs and variables. However, you will not be able to call the functions you delete. A tempting to do so will result in an FC error. The only way to restore a function that has been deleted is to retype BASIC.

The following is the dialog which will occur:

WANT SIN-COS-TAN-ATN? Answer "Y" to retain all four of the functions, "N" to delete all four, or "A" to delete ATN only.

Now BAS C W (YD 54)

XXXX BYTES FREE **XXXX"** is the number of bytes available for programs, variables, array storage and the stack. It does not include the OS kernel.

It will then print out the BASIC version and
COPYRIGHT MICROSOFT CO
with the year of the copyright, and finally
OK

You will now be ready to begin using KIM I BAS C

USING THE CASSETTE INTERFACE

To save a program on tape prepare the cassette just as though a dump command were about to be issued to the KIM-1 monitor. Then type "SAV". A label "O" of "P" is issued for all BASIC files, so only one BASIC program can be saved per tape. After completion of the "SAV" command control will be returned to the KIM-1 monitor. Reenter BASIC at the "RETSAV" location specified for your version using the "G" command.

[illegible]

The current version of BASIC does not support loading and saving of data files.

• KB-9 INTEGER VARIABLES

Fixed variables are allowed in the 9.0.0 version of MATLAB. They are global but fixed by the user. They are used for things that are in the variable set not from a floating point value. The amount of memory they take is added to the heap as they are created whereas the floating point variables are stored in the heap. They are not stored in the heap but are stored in the fixed memory space. They do not save time in fact, they are usually slower to use than floating point values.

Basic/machine Language interface

the 4 are the steps required to reach an end goal

set as default. The K5 versions of Windows 95 O2 B45 C and B45 G 100 are also similar. The B45 C is used in programs made by the highest 16-bit compilers and may be able to use the B45 C value 1000. However, the higher 16-bit compilers used the B45 C value 1000. The higher 16-bit compilers add 1000 to the value 1000 and reading values from memory.

A machine language routine must not be in a memory page used by BASIC so that it is not

- Below 2000 hex but above 200 hex or
 - Above the decimal address typed into "memory size" prompt
 - Non-contiguous with the RAM at 2000 hex
- Store the routine into memory. This can be done either before or after BAS-IC is loaded. The KIM cassette had an assembler keying into memory or BAS-IC's POKE command may be used.
- Notify BAS-IC of the change in memory. The `USRLOC` which is 2040 hex may be used. The `M-BAS-IC` may be POKE'd into the address of the `USR` routine (address is the 2040 hex B256 decimal) or be given the low 8-bits of the address and 2041 hex invoking the "USR" function before modifying `USRLOC` will cause an "ILLEGAL QUANTITY" error since the original contents of `USRLOC` contain the address of the "ILLEGAL QUANTITY" error routine.
- The machine language routine must be loaded. The "USR" function is used for this purpose. A single numeric value must be given as the argument of `USR`. BAS-IC will search for the address contained in `USRLOC`. The `USR` routine may modify all or the registers. An `END`

Da a can be passed to the machine language routine in two ways:

String variables also use one byte of string space for each character in the string. This is true whether the string variable is a simple string variable like A\$(1) or an element of a string matrix such as Q\$(5,3).

When a new function is defined by a DEF statement, 6 bytes are used to store the definition.

Reserved words such as END, PRINT, IF, NEXT, and the names of the intrinsic functions such as COS, INT and STR\$ take up only one byte of program space. All other variables in programs use one byte of program space each.

When a program is being executed, space is dynamically allocated on the stack as follows:

- 1) Each active FOR...NEXT loop uses 22 bytes.
- 2) Each active GOSUB (one that has not returned yet) uses 6 bytes.
- 3) Each parenthesis encountered in an expression uses 4 bytes and each temporary result calculated in an expression uses 12 bytes.

SPEED HINTS

The hints below should improve the execution time of your BASIC program. Note that some of these hints are the same as those used to describe the space used by your programs. This means that in many cases you can increase the efficiency of both the speed and size of your programs at the same time.

Delete all unnecessary spaces and REM's from the program. This may cause a small decrease in execution time because BASIC would otherwise have to ignore or skip over spaces and REM statements.

2) **THIS IS PROBABLY THE MOST IMPORTANT SPEED HINT BY A FACTOR OF 10.** Use variables instead of constants. It takes more time to evaluate the value of a constant than it does to evaluate the value of a simple 0 or 1 variable. This is especially true for arrays which are used in loops where the code that is executed repeats often.

3) Variables which are encountered during the execution of a BASIC program are allocated in the stack in the order the variable is used. This means that a statement such as `A=B, B=A, C=A`, will place A first, B second, and C third in the symbol table (assuming line 5 is the first taken or executed in the program). Later in the program when BASIC finds a reference to the variable A, it will seek through the entry in the symbol table to find A, two entries to find B and three entries to find C, etc.

4) NEXT statements with an index variable. NEXT is somewhat faster than NEXT I because no check is made to see if the variable specified in the NEXT is the same as the variable in the next statement.

CONVERTING BASIC PROGRAMS NOT WRITTEN FOR THE KIM-1

Though implementations of BASIC tend to be so compact and in many ways similar, there are some incompatibilities which you should watch for when you are planning to convert some BASIC programs to run on the KIM-1.

1. Array subscripts. Some BASICs use "[" and "] " to denote array subscripts. KIM-1 BASIC uses "(" and ")".
2. Strings. A number of BASICs force you to dimension arrays by length in bytes before use. You should always add dimension statements to the code from the program that you wish to convert. A declaration in the form `DIM A$(I,J)` declares a string matrix of J elements each of which has a length I. Convert DIM statements of this type to equivalent ones in KIM-1 BASIC: `DIM A$(J)`.

KIM-1 BASIC uses "&" for string concatenation not "+", "or" "&".

KIM-1 BASIC uses LEFT\$, RIGHT\$ and MID\$ to take substrings of strings. Other BASICs use A\$(I) to access the Ith character of the string A\$, and A\$(I,J) to take a substring of A\$ from character position I to character position J. Convert as follows:

OLD	NEW
A\$(I)	MID\$(A\$,I,1)
A\$(I,J)	MID\$(A\$,I,J+1)

This assumes that the reference to a substring of A\$ is in an expression or is on the right side of an assignment. If the reference to A\$ is in the left hand side of an assignment and X\$ is the string expression used to replace characters in A\$, convert as follows:

OLD	NEW
A\$(I)=X\$	A\$=LEFT\$(A\$,I-1)+X\$+MID\$(A\$,I+1)
A\$(I,J)=X\$	A\$=LEFT\$(A\$,I-1)+X\$+MID\$(A\$,J+1)

3. Multiple assignments. Some BASICs allow statements of the form: `500 LET B=C=0`. This statement would set the variables B & C to zero.

In KIM-1 BASIC this has an entirely different effect. All the variables on the right of the first one would be initialized as local comparison addresses. This would set the variable B to 1 if C equaled 0. If C did not equal 0, B would be set to 0. The easiest way to convert statements like this is to rewrite them as follows:

`500 C=0: B=C`

- 4) Some BASICs use ";" instead of ":" to delimit multiple statements per line. Change the ";" to ":" in the program.
- 5) Paper tapes punched by other BASICs may have strings at the end of each line instead of the three per line recommended for use with KIM-1 BASIC.

To go around this, try to use the tape feed or stop in the Teletype to stop the tape from reading as soon as KIM-1 BASIC types a carriage return at the end of the line. Wait a second, and then continue feeding in the tape.

When you have finished reading in the paper tape of the program, be sure to punch a new tape in KIM-1 BASIC's format. This will save you from having to repeat this process a second time.

- 6) Programs which use the MAT functions available in some BASICs will have to be rewritten using FOR...NEXT loops to perform the appropriate operations.

DERIVED FUNCTIONS

The following functions, while not intrinsic to KIM-1 BASIC, can be calculated using the existing BASIC functions.

FUNCTION	FUNCTION EXPRESSED IN TERMS OF BASIC FUNCTIONS
SECANT	SEC(X) = 1/COS(X)
COSECANT	CSC(X) = 1/SIN(X)
COTANGENT	COT(X) = 1/TAN(X)
INVERSE SINE	ARCSIN(X) = ATN(X/SQR(1-X*X+1))

INVERSE COSINE
 INVERSE SECANT
 INVERSE COSECANT
 INVERSE COTANGENT
 HYPERBOLIC SINE
 HYPERBOLIC COSINE
 HYPERBOLIC TANGENT
 HYPERBOLIC SECANT
 HYPERBOLIC COSECANT
 HYPERBOLIC COTANGENT
 INVERSE HYPERBOLIC
 SINE
 INVERSE HYPERBOLIC
 COSINE
 INVERSE HYPERBOLIC
 TANGENT
 INVERSE HYPERBOLIC
 SECANT
 INVERSE HYPERBOLIC
 COSECANT
 INVERSE HYPERBOLIC
 COTANGENT

$\text{ARCCOS}(X) = \text{ATN}(X \text{ SQR}(X^2 - 1)) + 1.5708$
 $\text{ARCSEC}(X) = \text{ATN}(\text{SQR}(X^2 - 1)) + (\text{SGN}(X - 1) * 1.5708)$
 $\text{ARCCSC}(X) = \text{ATN}(1/\text{SQR}(X^2 - 1)) + (\text{SGN}(X - 1) * 1.5708)$
 $\text{ARCCOT}(X) = \text{ATN}(X) + 1.5708$
 $\text{SINH}(X) = (\text{EXP}(X) - \text{EXP}(-X)) / 2$
 $\text{COSH}(X) = (\text{EXP}(X) + \text{EXP}(-X)) / 2$
 $\text{TANH}(X) = (\text{EXP}(X) - \text{EXP}(-X)) / (\text{EXP}(X) + \text{EXP}(-X))$
 $\text{SECH}(X) = 2 / (\text{EXP}(X) + \text{EXP}(-X))$
 $\text{CSCH}(X) = 2 / (\text{EXP}(X) - \text{EXP}(-X))$
 $\text{COTH}(X) = (\text{EXP}(X) + \text{EXP}(-X)) / (\text{EXP}(X) - \text{EXP}(-X))$

 $\text{ARGSINH}(X) = \text{LOG}(X + \text{SQR}(X^2 + 1))$

 $\text{ARGCOSH}(X) = \text{LOG}(X + \text{SQR}(X^2 - 1))$

 $\text{ARGTANH}(X) = \text{LOG}((1 + X) / (1 - X)) / 2$

 $\text{ARGSECH}(X) = \text{LOG}((\text{SQR}(X^2 + 1) + 1) / X)$

 $\text{ARGSCH}(X) = \text{LOG}((\text{SGN}(X) * \text{SQR}(X^2 + 1) + 1) / X)$

 $\text{ARGCOTH}(X) = \text{LOG}((X + 1) / (X - 1)) / 2$

IMPORTANT LOCATIONS (HEX)

	<u>KB-9P</u>	<u>KB-9</u>
LOWEST LOCATION	2000	2000
HIGHEST LOCATION	3FFF	4260
INIT (START)	3E91	4065
GLOAD (RETURN FROM GOOD LOAD)	2278	0000
BDLOAD (RETURN FROM BAD LOAD-ALSO PATCH LOCATIONS 1 & 2)	2455	2523
RETSAY (RETURN FROM SAVE)	26A2	275F
USRLOC (FOR USR ADDRESS)	000B	2040
AYINT (PASS ARGUMENT OF USR FUNCTION TO THE ZERO PAGE ADDRESS REFER TO ZERO PAGE LISTING)	2F04	2FC2
GIVAYF (RETURN A,Y FROM MACHINE CODE USR ROUTINE)	30D7	3195
ISCNTC (CHECK FOR CONTROL/C)	2610	26DA
LOCATION OF CALL TO KIM-1 INPUT ROUTINE	238C	2456
LOCATION OF CALL TO KIM-1 OUTPUT ROUTINE	2991	2A51

APPLICATION NOTE: Microsoft has advised us that the value of .1 cannot be represented exactly in binary floating point. Programs magnify this inaccuracy until it shows up on print-out. This can be handled by:

- (1) Always use integers which are exact and then scale. As an example, 14 X 1.23 would be (14*123) / 10.
- (2) Use tests such as ABS(X-Z) < 1E-6 or STR\$(X)=STR\$(Z)
- (3) Round out to the number of digits you want before printing.

The way to save data using Microsoft KIM BASIC is to:

- (1) Have it in DATA statements and use CSAVE.
- (2) Write your own USR function to perform this.

Microsoft will provide access to cassette data in a future release if there is enough general interest. Drop us a card if you are interested.

If BASIC is waiting for an input and you type a carriage return, you will exit your program and return to the immediate Mode of BASIC. This is intentional. You can return to the program using CONT. If you want to prevent accidentally leaving the program in this manner, you can use POKE or the KIM Monitor to change the A5 to an A9 at location 10920 decimal (2AA6 HEX) for the RAM version. Now, however, you will only be able to exit BASIC by using CONTROL C, encountering a STOP in the program, or if the program comes to an END.

SOFTWARE: Three KIM-1 routines have been written by Ralph Bugg, a user, which are compatible with the KIM-1 9 digit BASIC. These routines are as follows.

BUGGWARE #1. A video driver making the Kent-Moore video board Catalog 60083 (32 X 16) compatible with KIM BASIC.

A routine for operating KIM BASIC from a parallel input keyboard
\$4.00 postage paid

A routine for output to a Baudot (5 level) Teletype
\$4.00 postage paid

BUGGWARE #2. A video driver making the Kent-Moore video board catalog 60117 (64 X 16)

From time to time we will make a mailing of miscellaneous information gathered on the KIM BASIC and related items. If you have any tips you would like to share with others, please send them in. At this point, there is no schedule or promise for the next issue but it will be sent out when enough information is accumulated to make it worthwhile.

HYPERTAPE TIP: For those of you who are unaware of Hypertape, this is a program written by Jim Butterfield of Toronto, Canada and published in Eric Rehnke's KIM-1 User Notes, Volume 1, Issue 2. Hypertape allows you to record on cassette in 1/6 the time of standard KIM dump routine. For parallel input Ralph Bugg suggests you place Hypertape at 0300 and enter the following changes:

	<u>KB-9P</u>	<u>KB-9</u>			
	035F	035F	60		
	269F	275C	20	00	03
Per Chuck Kingston, this may be done using the KIM serial port with:					
	035E	4C	C2	03	03C5 58
	03C2	20	8C	1E	03C6 60

Following is information for reading a line:

- (1) The routine to input a line from the terminal is located at:

KB 9.....2420-2455 (starts at 2426)
KB9P.....2356-239B (start at 235C)

- (2) The compare for a (line delete) is at:

KB 9.....243B
KB9P..... 2371

- (3) The compare for a (character delete) is at location:

KB-9.....243F
KB9P.....2375

(4) Note that codes < 20 HEX and ≥ 70 HEX have already been ignored by the compare above these.

If you are having any problems with your KIM-1 BASIC by Microsoft, please write to us. If at all possible, include a print-out illustrating your difficulty. Document your print-out with handwritten notes indicating the difficulty.

KIM-1 digit BASIC by Microsoft is also available in a version adapted for the TIM monitor manufactured by MOS Technology. TIM stands for Teletype Input Monitor and is often used by people developing a CPL of their own design. The TIM is part #6510-034 and sells for \$14.15. The manual for application of the TIM sells for \$4.95. Both items are available from Johnson Computer.

You may purchase usage rights or source code listings from Microsoft as follows

IN-HOUSE OBJECT CODE RIGHTS for use on a specific project are available for \$750.00. This would be appropriate for a user who would otherwise have to purchase several copies of BASIC authorized for use on a specific KIM unit. As an example, using KIM in an in-house test stand and then building ten more test stands, each using a KIM.

IN-HOUSE SOURCE RIGHTS to all versions of 6502 BASIC is available to customers who would like to customize the program for each KIM-1. The price is \$3,000.00 for the in-house for each use of the source code.

OEM SOURCE CODE RIGHTS are available for making Microsoft's 6502 BASIC a part of a product being manufactured for resale. The price to an OEM for source code rights is \$3,000.00 plus \$35.00 for each copy up to 1,000 copies (\$35,000.00) after which they then own the rights. This can be purchased outright for \$21,000.00. An OEM account receives the source listing and may sell the object code as a part of his product but the source code may not be sold, distributed, given out or leave the account facility.

OEM OBJECT CODE RIGHTS is priced at \$1000.00 plus \$35.00 per copy up to 1000 copies. Source code available at a later date for the 2000 difference.

JOHNSON COMPUTER

Implementation of a CONTROL C is difficult due to the nature of the KIM I/O port. Because only a bit at a time comes into the port, it is impossible to handle a character typed during computation. By the time BASIC tries to detect a CONTROL C character, several bits of data may already have passed through the port.

Exiting to the monitor and re-entering at the WARM START location will stop the program, however, the "CONT" command will not work properly. In fact, exiting to monitor can leave BASIC in a state where all variable accesses hang the machine until "RUN", "NEW", or "CLEAR" is typed or a program line is changed. However, this is very unlikely.

Changing to a parallel input keyboard eliminates the above problem.

The CONTROL O facility can be handled by POKEing or using the KIM monitor to set the CONTROL O flag location of 0017 in KB-9P, or 0014 in KB-9, to FF for no output or to 00 for output.

P.O. BOX 523 MED NA, OHIO 44256

(216) 725-4560

INVERSE COSINE
 INVERSE SECANT
 INVERSE COSECANT
 INVERSE COTANGENT
 HYPERBOLIC SINE
 HYPERBOLIC COSINE
 HYPERBOLIC TANGENT
 HYPERBOLIC SECANT
 HYPERBOLIC COSECANT
 HYPERBOLIC COTANGENT
 INVERSE HYPERBOLIC
 SINE
 INVERSE HYPERBOLIC
 COSINE
 INVERSE HYPERBOLIC
 TANGENT
 INVERSE HYPERBOLIC
 SECANT
 INVERSE HYPERBOLIC
 COSECANT
 INVERSE HYPERBOLIC
 COTANGENT

$\text{ARCCOS}(X) = \text{ATN}(X/\text{SQRT}(X^2+1)) + 1.5708$
 $\text{ARCSEC}(X) = \text{ATN}(\text{SQRT}(X^2-1)) + (\text{SGN}(X) * 1.5708)$
 $\text{ARCCSC}(X) = \text{ATN}(1/\text{SQRT}(X^2-1)) + (\text{SGN}(X) * 1.5708)$
 $\text{ARCCOT}(X) = \text{ATN}(X) + 1.5708$
 $\text{SINH}(X) = (\text{EXP}(X) - \text{EXP}(-X)) / 2$
 $\text{COSH}(X) = (\text{EXP}(X) + \text{EXP}(-X)) / 2$
 $\text{TANH}(X) = \text{EXP}(X) / (\text{EXP}(X) + \text{EXP}(-X))^{1/2}$
 $\text{SECH}(X) = 2 / (\text{EXP}(X) + \text{EXP}(-X))$
 $\text{CSCH}(X) = 2 / \text{EXP}(X) - \text{EXP}(-X)$
 $\text{COTH}(X) = \text{EXP}(X) / (\text{EXP}(X) + \text{EXP}(-X))^{1/2}$

 $\text{ARGSINH}(X) = \text{LOG}(X + \text{SQRT}(X^2+1))$

 $\text{ARGCOSH}(X) = \text{LOG}(X + \text{SQRT}(X^2-1))$

 $\text{ARGTANH}(X) = \text{LOG}((1+X)/(1-X))^{1/2}$

 $\text{ARGSECH}(X) = \text{LOG}(\text{SQRT}(X^2-1) + 1) - X$

 $\text{ARGSCH}(X) = \text{LOG}(\text{SGN}(X) * \text{SQRT}(X^2+1) + 1) - X$

 $\text{ARGCOTH}(X) = \text{LOG}((X+1)/(X-1))^{1/2}$

IMPORTANT LOCATIONS (HEX)

	<u>KB-9P</u> <u>(FROM)</u>	<u>KB-9</u>
LOWEST LOCATION	2000	2000
HIGHEST LOCATION	3FFF	4260
INIT (START)	3E91	4065
CLOAD (RETURN FROM GOOD LOAD)	227B	0000
BDLOAD (RETURN FROM BAD LOAD-ALSO PATCH LOCATIONS 1 & 2)	2455	252
RETSAV (RETURN FROM SAVE)	26A2	275F
USRLOC (FOR USER ADDRESS)	000B	2040

AYINT (PASS ARGUMENT OF LSR FUNCTION TO THE ZERO PAGE ADDRESS See Zero Page Listing)	2F04	2FC2
GIVAYF (RETURN A,Y FROM MACHINE CODE LSR ROUTINE)	30D7	3195
ISCNTC (CHECK FOR CONTROL/C)	2610	26DA
LOCATION OF CALL TO KIM-1 INPUT ROUTINE	238C	2456
LOCATION OF CALL TO KIM-1 OUTPUT ROUTINE	2991	2A51

Page 0 is configured as follows:

NOTICE: The PROM version initializes at 3E91. It does not provide SIN, COS, TAN, ATN, etc. Memory used is \$2000 to \$1FFF. Workspace begins at \$4000.

KB-9 uses all locations 0-EC and FF

PROM used all locations 0-DF and FF

PROM	RAM	
KB-9P	KB-9	DESCRIPTION
227B	0	JMP to WARM START BASIC.
6	6	Address of routine to transfer USR argument to y,A (AYINT)
8	8	Address of routine to transfer (y,A) to result of USR function (GIVAYF).
13	14	FLAG set to FF if output is suppressed (CONTROL 0 mode). Set to 0 otherwise.
14	15	Number of NULLS to print.
15	16	Current terminal column (equal to POS (0)).
16	17	Line length.
17	1E	Position beyond which there are no more comma fields Equal to 14*(INT (line length/14)-1).
1A	1B	Input buffer .72 decimal bytes.
76	78	Pointer to start of program.
78	7A	Pointer to start of simple variable table
7A	7C	Pointer to start of array table.
7C	7E	First location unused by array table.
7E	80	Lowest location used by string data.
82	84	Highest memory location in use by BASIC.
84	86	Current line number.
A9	AE	Floating accumulator
B4	B1	Value of argument in AYINT
B9	CD	Routine to read a character from current program position
D1	D8	Current random number.
D5	DD	First unused page 0 location.
FF	FE	Used by STR\$ function.

JOHNSON COMPUTER

NOTICE

For perfect load and dump operations using a cassette the following should be observed:

1. The small magnetic bar located on the polished surface of the recording head used in the tape recorder should be aligned to the magnetic information recorded on the tape. Typically the recording head is secured in place with two screws. One screw is spring loaded and allows the recording head to be shifted slightly for alignment with the information stored on the tape. It is possible the recording head was not aligned properly at the factory or the head has been jarred to an improper setting by rough handling of the recorder itself. If a production type tape has been recorded properly it can be used to find the proper setting of your recording head. While playing the tape the spring loaded recording head adjustment screw can be varied to obtain maximum brilliance (sharpness) of tone. Additionally, proper adjustment will allow programs to be stored on either side (track) of the cassette with no interference between the two. Recording head misalignment is one of the chief causes of improper loading from the cassette in the KIM-1 system.
2. If, after verifying proper recording head alignment, you still have problems, check the setting of VR-1 (the SK potentiometer) located to the left of the keyboard. Connect a jumper from terminal P on the application connector to terminal L on the application connector. Next, connect a DC volt meter between terminal X on the expansion connector and ground. Then, referring to the KIM-1 User's manual Page E2 adjust the potentiometer to get a reading of +1.4V (no less than .7V and no more than 3.0V). This is a very touchy adjustment. If you have properly aligned your recorder head and the KIM-1 is properly calibrated you should have no problem dumping to your recorder and loading from your recorder. Should you still have a problem you might check to be sure that the tape you are trying to load in your KIM-1 was recorded on a properly adjusted recorder head. If this was not done, the program can be saved by readjusting your recorder head in the wrong position so that it properly lines up with the bad cassette, loading to KIM and then readjust your head to its proper setting and re-recording the program from your KIM to a clean cassette.

Your KIM-1 Basic by Microsoft cassette has been recorded using HYPERTAPE developed by Jim Butterfield of Toronto, Canada and published in KIM-1 Users Notes, Issue 2, Pages 12, 13 and 14. This program enables you to record and play back in 1/6th of the time required using the standard KIM-1 tape routines. Playing back into your KIM-1 from a HYPERTAPE recording follows the same procedure as the standard KIM-1 tape loading routine using 1873 and GO. HYPERTAPE is more sensitive to the adjustment of the tape recorder head. If you do not get a load on the first try then there is a strong possibility that your tape recorder head has not been aligned exactly the same as ours. You can "tune in" to the tape by using some delightful programs published in KIM-1 Users Notes.

1. Before changing the setting of your tape recorder head take one of your own cassettes and generate a SYNC STREAM on your own recorder using the program from Page 11 of KIM-1 Users Notes, Volume 1, Issue 2.
2. Now, load the VUTAPE into your KIM-1. VUTAPE was written by Jim Butterfield of Toronto, Canada and published in KIM-1 Users Notes, Issue 2, Page 12.
3. After loading VUTAPE and 0000, press GO. The last character in your display will come on in a random fashion. Now go to your KIM BASIC cassette and locate the 30 second SYNC STREAM (you can tell it by the steady sound) which we have recorded immediately following the end of the KIM-1 BASIC. This is at about 3 minutes and 15 seconds into the tape. Adjust your volume control to about 2. While playing this sync stream into your KIM-1, adjust your tape recorder head set screw so that the sync pulses "lock in" on the right end of display. This adjustment should be very close to the original adjustment of your head. Once locked in, lower the volume control to about 1 or even .5 and adjust again.
4. Next you might want to check your phase lock loop (VR-1) adjustment on your KIM-1. This is easiest done by using the PLL SET program by Louis Edwards, Jr. of Trenton, N.J. and published on Page 3 of KIM-1 Users Notes Issue 5. Now go to the beginning of the KIM-1 BASIC cassette, set up your load routine. Enter Ident 01 at 17F9. Be sure 00F1 is loaded with 00. Go to 1873, set volume control at about 8 and GO. You should load in three minutes. Now refer to the enclosed documentation for BASIC operating instructions.

KIM-1 User Notes is published by Erick Rehnke, 109 Center Street, West Norriton, PA. 19401. Subscriptions are 6 issues for \$5.00 in U.S.A. and Canada, \$10.00 for 6 issues elsewhere. Johnson Computer will accept subscriptions on purchase orders. Individuals make checks payable to "KIM-1 Users Notes". Also, "The First Book of KIM", by ORB (Ocker, Rehnke & Butterfield) has reprinted most important programs published in issues 1 through 5 and also includes excellent text on KIM and the 6502. Price is \$9.50 available through Johnson Computer, P.O. Box 523, Medina, Ohio 44256.

SYNC STREAM - 0000 A0 BF 8C 43 17 A9 16 20 7A 19 D0 F9

VUTAPE - 0000 D8 A9 7F 8D 41 17 A9 13 85 E0 8D 42 17 20 41 1A 46 F9 05 F9 85 F9
 0016 8D 40 17 C9 16 D0 E9 20 24 1A C9 2A D0 F5 A9 00 8D E9 17 20 24 1A
 002C 20 00 1A D0 D5 A6 E0 E8 E8 E0 15 D0 02 A2 09 86 E0 8E 42 17 AA
 0041 BD E7 1F 8D 40 17 D0 DB

PLL SET - 1780 A9 07 8D 42 17 A9 01 8D 01 17 85 E1 A9 7F 8D 41 17 A2 09 A0 07
 1795 2C 42 17 30 02 A0 38 8C 40 17 3E 42 17 2C 47 17 10 FB E6 E2 30 04
 17AB A9 91 D0 03 A9 93 EA 8D 44 17 A9 01 45 E1 85 E1 BD 00 17 E8 E8
 17CD E0 15 D0 CF F0 CB

JOHNSON COMPUTER

BULLETIN

TO: All KIM BASIC users.

To more specifically describe the start-up procedure for loading the KIM BASIC from cassette, we are offering the following:

- (1) Have at least 16K of memory added on to your KIM-1, beginning at address 2000 HEX.
- (2) We suggest you make a memory test at this point to be sure you have no bad memory bits.
- (3) If you are loading via the KIM-1 hexadecimal keyboard, enter the following:

- AD 00F1 DA 00
AD 17F9 DA 01
+ 00 + 1C + 00 + 1C + 00 + 1C
AD 1873 G
- Start tape recorder with volume set at approximately 80% and treble on full.
- After three minutes, you should get a display of 0000.
- Flip your switch to transfer to your terminal.
- Press "RUBOUT" key on your terminal.
- On your terminal, type in "4065" (starting address for KIM 9 digit BASIC) followed by the space bar.
- Type "G".
- KIM BASIC should come up asking memory size.
- Tap the "RETURN" key. However, if you want to set aside memory at the high end, answer with the decimal value of the highest memory location KIM BASIC can use. KIM BASIC memory must be contiguous from 2000 HEX up.
- KIM BASIC should now ask for terminal width.
- Tap the "RETURN" key if you have a 72 character terminal width. If you have less than 72 characters across, or if you have more than 72 characters and plan on using the entire terminal width, type in actual terminal width.

(Cont. on Page 2)

P.O. BOX 523 MEDINA, OHIO 44256

(216) 725-4560

- BASIC will ask if you want SINE, COSINE, etc. We suggest you always answer "Y", for YES.

You should now be in BASIC.

- (4) If you intend to transfer to the keyboard for the entire start-up, use the following procedure:

- Press "RS" on the KIM-1 hexadecimal keyboard.
- Press "RUBOUT" on your terminal keyboard.
- Type in the following on your terminal keyboard:

```
00F1 (space bar) 00 .
17F9 (space bar) 01 .
          00 . 1C . 00 . 1C . 00 . 1C .
1873 (space bar) G
```

- At this point, turn on the cassette. After a successful load, the screen will display:

```
KIM
0000
```

- type starting address and routine as follows:

```
4065 (space bar) G
```

- Answer questions on memory size, terminal width and transcendental functions as on page one.

You are now in BASIC.

Saving programs on cassette is accomplished as follows:

- (1) Plan to save only one program per cassette. That program should be in your memory when you are ready to record. Use the following procedure:

- On your terminal keyboard, type "SAVE".
- Start your tape recorder. When leader has passed, type "G".
- A successful SAVE will be indicated by the following display on your screen:

```
KIM
0000
```

- (2) Since 0000 is WARM START, get back to BASIC simply by typing "G".

In order to load a program from cassette back into KIM, use the following procedure:

- Enter BASIC as above and then type "LOAD".